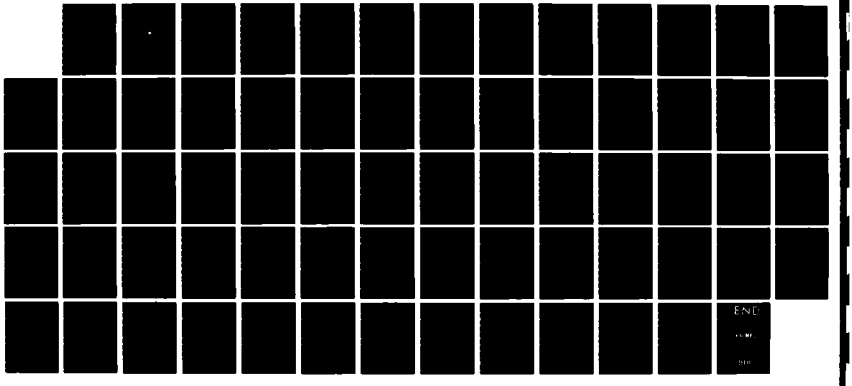
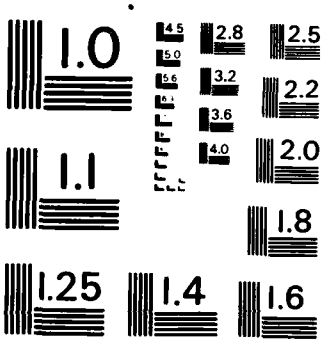
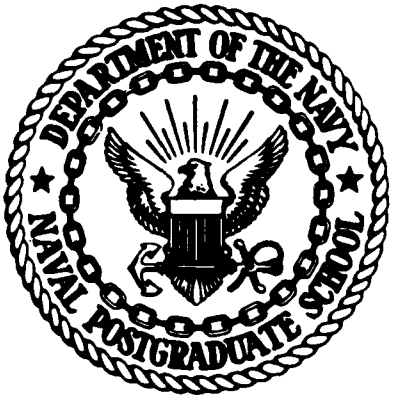
AD-A161 238 NAVY MILITARY STANDARDS FOR TECHNICAL SOFTWARE DOCUMENTATION OF EMBEDDED TACTICAL SYSTEMS; A CRITICAL REVIEW(U) NAVAL POSTGRADUATE SCHOOL MONTEREY CA 1/1

UNCLASSIFIED H C LYON SEP 85 F/G 9/4 NL





NAVAL POSTGRADUATE SCHOOL
Monterey, California



THESIS

NAVY MILITARY STANDARDS FOR
TECHNICAL SOFTWARE DOCUMENTATION OF
EMBEDDED TACTICAL SYSTEMS;
A CRITICAL REVIEW

by

Harvey Channing Lyon

September 1985

Thesis Advisor:	Gordon H. Bradley
Co-advisor:	Carl R. Jones

Approved for public release; distribution unlimited

UNCLASSIFIED

SECURITY CLASSIFICATION OF THIS PAGE *(When Data Entered)*

REPORT DOCUMENTATION PAGE		READ INSTRUCTIONS BEFORE COMPLETING FORM
1. REPORT NUMBER	2. GOVT ACCESSION NO. AD-A161 238	3. RECIPIENT'S CATALOG NUMBER
4. TITLE *(and Subtitle)* Navy Military Standards for Technical Software Documentation of Embedded Tactical Systems; A Critical Review		5. TYPE OF REPORT & PERIOD COVERED Master's Thesis September 1985
		6. PERFORMING ORG. REPORT NUMBER
7. AUTHOR(s) Harvey Channing Lyon		8. CONTRACT OR GRANT NUMBER(s)
9. PERFORMING ORGANIZATION NAME AND ADDRESS Naval Postgraduate School Monterey, CA 93943-5100		10. PROGRAM ELEMENT, PROJECT, TASK AREA & WORK UNIT NUMBERS
11. CONTROLLING OFFICE NAME AND ADDRESS Naval Postgraduate School Monterey, CA 93943-5100		12. REPORT DATE September 1985
		13. NUMBER OF PAGES 65
14. MONITORING AGENCY NAME & ADDRESS(if different from Controlling Office)		15. SECURITY CLASS. (of this report)
		15a. DECLASSIFICATION/ DOWNGRADING SCHEDULE

16. DISTRIBUTION STATEMENT *(of this Report)*

Approved for public release; distribution is unlimited

17. DISTRIBUTION STATEMENT *(of the abstract entered in Block 20, if different from Report)*

18. SUPPLEMENTARY NOTES

19. KEY WORDS *(Continue on reverse side if necessary and identify by block number)*

Software Documentation
Documentation
DOD-STD-1679A(Navy)
Software Life-cycle

20. ABSTRACT *(Continue on reverse side if necessary and identify by block number)*

This thesis critically reviews the Navy's embedded tactical software development methodology as defined in DOD-STD-1679A(Navy). The emphasis of the thesis is on the documentation produced as a result of following that methodology. Both the development methodology, and the documentation produced are compared to management and content recommendations provided by the National Bureau

(continued)

DD FORM 1 JAN 73 1473 EDITION OF 1 NOV 65 IS OBSOLETE

S N 0102-LF-014-6601

UNCLASSIFIED

SECURITY CLASSIFICATION OF THIS PAGE *(When Data Entered)*

of Standards and academic/commercial publications. The conclusion reached is that DOD-STD-1679A(Navy) is adequate for its purpose. However problems in documentation develop as a result of management's misinterpretation of the phased life-cycle development methodology described in DOD-STD-1679A(Navy) and the importance of a continous documentation effort.

Approved for public release; distribution is unlimited.

Navy Military Standards for Technical Software Documentation of Embedded Tactical Systems; A Critical Review

by

Harvey Channing Lyon
Lieutenant, United States Navy
B.S., United States Naval Academy, 1979

Submitted in partial fulfilment of the requirements for the degree of

MASTER OF SCIENCE IN INFORMATION SYSTEMS

from the

NAVAL POSTGRADUATE SCHOOL
September 1985

Author: Harvey Channing Lyon
Harvey Channing Lyon

Approved by:
Gordon H. Bradley, Thesis Co-Advisor

Carl R. Jones, Thesis Co-Advisor

Willis R. Greer, Chairman of Administrative Sciences

K.T. Marshall
Kneale T. Marshall, Dean of Information and Policy Sciences

ABSTRACT

This thesis critically reviews the Navy's embedded tactical software development methodology as defined in DOD-STD-1679A(Navy). The emphasis of the thesis is on the documentation produced as a result of following that methodology. Both the development methodology, and the documentation produced are compared to management and content recommendations provided by the National Bureau of Standards and academic/commercial publications. The conclusion reached is that DOD-STD-1679A(Navy) is adequate for its purpose. However problems in documentation develop as a result of management's misinterpretation of the phased life-cycle development methodology described in DOD-STD-1679A(Navy) and the importance of a continous documentation effort.

TABLE OF CONTENTS

I. INTRODUCTION ---------- 7

II. SOFTWARE DOCUMENTATION, WHAT IS IT? ---------- 12

A. THE SOFTWARE LIFE-CYCLE ---------- 12

B. DOCUMENTATION WITHIN THE SOFTWARE LIFE-CYCLE -- 19

C. TECHNICAL DOCUMENTATION GOALS ---------- 23

III. NAVY TECHNICAL DOCUMENTATION ---------- 26

A. THE NAVY TACTICAL SOFTWARE LIFE-CYCLE --------- 26

B. NAVAL TECHNICAL SOFTWARE DOCUMENTATION REQUIREMENTS ---------- 31

IV. NAVAL DOCUMENTATION, PROBLEMS AND SOLUTIONS ------- 48

A. NAVAL DOCUMENTATION, HOW DOES IT STACK UP ----- 48

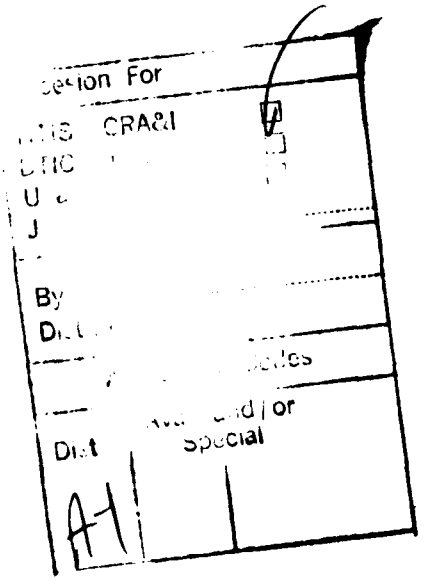
B. NAVAL DOCUMENTATION, THE PROBLEM AND AND CAUSES ---------- 51

C. DOCUMENTATION, A SOLUTION IN ITSELF ---------- 54

V. SUMMARY OF CONCLUSIONS ---------- 60

LIST OF REFERENCES ---------- 64

INITIAL DISTRIBUTION LIST ---------- 65



ACKNOWLEDGEMENTS

I wish to thank Dr. Gordon Bradley and Dr. Carl Jones of the Naval Postgraduate School for their direction and assistance in writing this thesis. Also I wish to thank Jim Raegan of the Naval Surface Weapons Center for his time, instruction, and enthusiasm in discussing the problems he has experienced in developing software for naval tactical systems. Additionally I wish to thank my father for his time spent in establishing contacts and collecting material for this thesis. There were many individuals that provided their time and insight for interviews without which this thesis would not have been possible. Although these individuals are too many to name, I wish to say thank you.

I. INTRODUCTION

The past ten years has seen much written on the improved methods of software development, top-down design, bottom-up testing, modular decomposition, structured programming, stepwise refinement and other related subjects. Even more recently, with the rising costs of maintenance, and the rapidly increasing functional requirements for the software being developed, the need for excellent documentation of software projects has become apparent with many articles addressing this issue. The most modern software development methodologies provide for this required documentation, addressing areas such as feasibility studies, requirements analysis, program performance specifications, test requirements, commented code, data flow diagrams, data dictionaries, interface specifications, and the like. A major purpose of these methodologies is to allow software developed by multiple people to stand independent of the individual. That is, to provide a method that captures the process used and the product produced by a group of individuals in a manner such that another group of individuals may understand the product produced and the process used to produce it without requiring further communication with the original group.

The Navy, with its extremely automated combat and engineering systems is like any other software producing/consuming firm and is one of the largest in the category of tactical real-time systems and engineering control systems. With software development and maintenance costs of naval tactical systems totaling in the billions of dollars annually, even a relatively small improvement in software development efficiency has an immense potential for a significant reduction in the cost of a system's development. As a consequence, the Navy has continually updated their Military Standards for Software Development (DOD-STD-1679A).

This thesis compares the recommended software development methodologies of the commercial and academic fields, and those recommended by the National Bureau of Standards with those presently in use by the Navy. The main emphasis is on the documents generated in these methodologies and the documentation process.

In this thesis the term "software" will be used as defined by Fairley [Ref. 1:p. 6] to mean the source code and all the associated documents and documentation that constitute the software product. Requirements documents, design specifications, source code, test plans, quality assurance and configuration management procedures, software trouble reports, etc., all constitute components of the software product and are included in the term "software".

"Documentation" will be defined as a description of the characteristics of an entity or process recorded for the purpose of transferring information about that entity or process. For the purposes of this thesis "documentation" will also include descriptions of intent or requirements such as the information within a computer program development plan, or a program requirements specification.

The term "documentation process" will refer to the methodology used to collect and explain the associated characteristics.

The term "documentation level", or level of documentation will refer to the amount of detailed information the documentation records about the entity or process in relation to the total amount of information that ever was available about the entity or process. The more information recorded in the documentation, the higher the level of the documentation.

"Document" will refer to the instrument used to transport the documentation from one individual to another, or from one group of individuals to another. Documents are produced to provide a medium with which documentation can be transferred.

In the process of writing this thesis, interviews were conducted with various key individuals experienced in Navy surface tactical embedded computer system design projects. These interviews were non-statistical in nature. The purpose

of these interviews was to provide experienced opinions and feelings on problems and causes evident in past and present development efforts. Additionally, this thesis only views those practices in use in the Naval Sea Systems Command, other commands within the Navy may have different definitions and methodologies. This thesis is not meant to apply to those commands.

The DDG-51 (AEGIS class destroyer) combat system design effort was used as the example software development project. This particular project was chosen because it reflects the efforts of an experienced design team (many of the team members were associated with the CG-47 design effort), and has adequate funding. Additionally, this project is relatively new (the computer program development plan was printed in January 1985) and had the opportunity to take advantage of the latest software design methodologies.

Chapter II presents suggested documentation requirements of the National Bureau of Standards and introduces a typical software life-cycle. Chapter III explains the Navy's tactical software life-cycle and the documentation requirements of DOD-STD-1679A (Navy). Chapter IV compares the Navy's standards with those recommended in the commercial/academic field and attempts to offer solutions to documentation problems expressed to be evident in Naval tactical software development. Chapter V summarizes th

conclusions reached in a method that is meant to serve a possible "tear out" summary requirement.

II. SOFTWARE DOCUMENTATION, WHAT IS IT?

This section defines "software engineering" as used in this thesis and introduces the software life-cycle concept by describing three models sometimes used to represent different concepts of the software life-cycle. This section also presents some document and documentation recommendations through the use of the life-cycle models they are associated with, and presents the goals or reasoning behind the technical documentation process and the documents produced.

A. THE SOFTWARE LIFE-CYCLE

Barry Boehm [Ref. 2:p. 16] defines software engineering as "the application of science and mathematics by which the capabilities of computer equipment are made useful to man via computer programs, procedures, and associated documentation". Fairley [Ref. 1:p. 2] defines software engineering as "the technological and managerial discipline concerned with systematic production and maintenance of software products that are developed and modified on time and within cost estimates".

Within the context of this thesis it is sufficient to define software engineering to be the technological and managerial discipline concerned with the systematic

production and maintenance of software keeping in mind that software, as defined earlier, includes the source code and all associated documents and documentation that constitute the software product.

Experience in software engineering has taught us that it is extremely important at the start of every software project to develop a model of the life-cycle of the particular software product. As Fairley states [Ref. 1:p. 37]: "A life-cycle model that is understood and accepted by all concerned parties improves project communication and enhances project manageability, resource allocation, cost control, and project quality."

The model must be developed specifically for the project at hand, however many generic models are available, that with minor modifications are usually well suited for the software project.

Perhaps the most basic model and the most traditional is the phased life-cycle model often described with the waterfall chart of Figure 1. Introduced in the early 70's, but conceptually existant in the mid 60's [Ref. 2:pp. 35-38], the phased life-cycle model segments the life cycle into a series of successive steps or phases. Each phase requires a well defined input, utilizes a well defined process, and results in a well defined output that is used as the starting point for the subsequent phase.

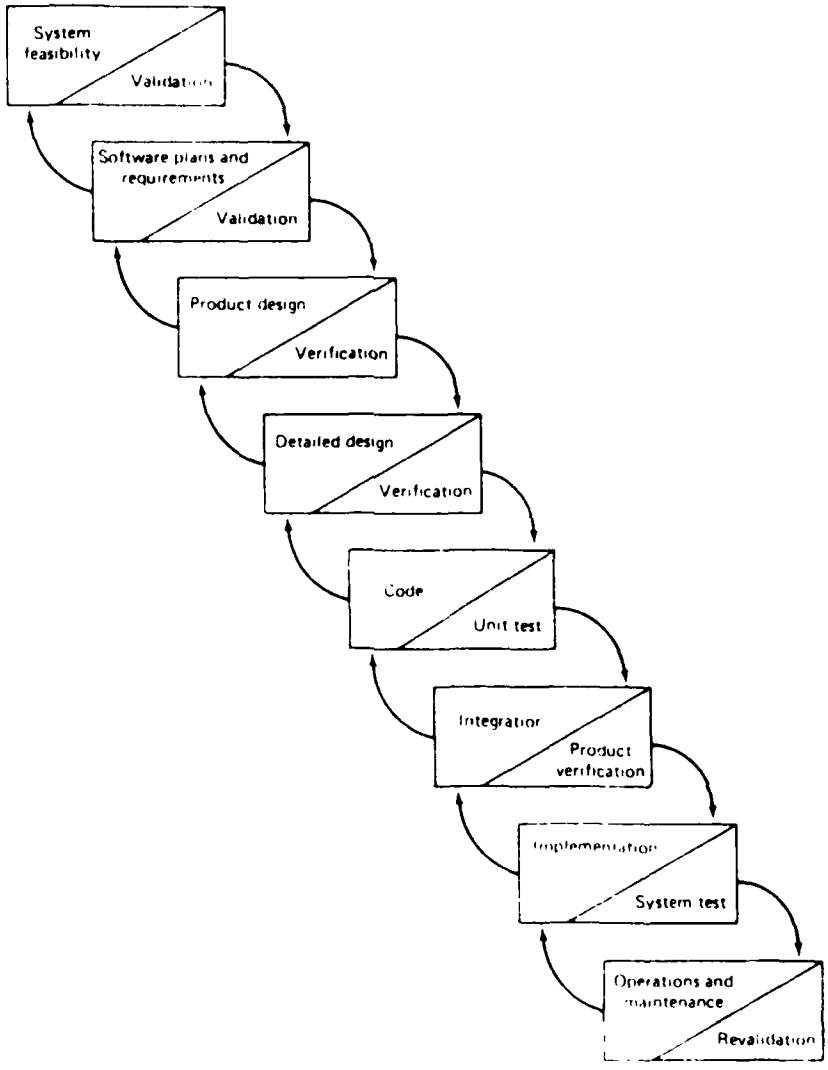


Figure 1: Waterfall Model of the Software Life-cycle [Ref. 2:p. 36]

The various phases of the phased life-cycle model are:

1. System Feasibility. Defining a preferred concept for the software product, and determining its life-cycle feasibility and superiority to alternative concepts.

2. Software Plans and Requirements. A complete, validated specification of the required functions, interfaces, and performance standards for the software product.

3. Product Design. A complete, verified specification of the overall hardware-software architecture, control structure, and data structure for the product, along with such other necessary components such as draft user's manuals and test plans.

4. Detailed Design. A complex, verified specification of the control structure, data structures, interface relations, sizing, key algorithms, and assumptions for each program component.

5. Coding. A complete, verified set of the program components.

6. Integration. A properly functioning software product composed of the software components.

7. Implementation. A fully functioning operational hardware-software system, including such objectives as program and data conversions, installation and training.

8. Maintenance. A fully functioning update of the hardware-software system. This subgoal is repeated for each update/ modification.

An important aspect of the phased life-cycle model is that each phase ends with verification or validation. Validation as it is used here means to make a dedicated effort to ensure the product of the phase is actually what was intended to be produced at the beginning of the phase. Informally, validation is "Are we building the right product?". Verification as it is used here refers to a dedicated effort to ensure that the product or output of the phase is correct for the input of the phase; Informally, verification is "Are we building the product right?". In a development design such as the phased life-cycle model that relies strongly on the output from one phase as the input to

the next phase, the determination of the correctness of the output before it is used as input is vital to the process. Verification and validation are planned conscientous efforts to eliminate errors within the development process.

There are many critics of the phased life-cycle approach. Among their complaints is that the approach does not accurately reflect the actual software development process, that it does not reflect the interaction and overlap between phases [Ref. 1:p. 41]. Nor does the phased life-cycle approach provide for prototypes, or enhancement methods. Additionally, if an error is made in the early stages, and is missed in the original validation, the error will not be evident until the final validation is made after the system is implemented. The phased life-cycle approach does not provide a means to alter a project's design once the implementation phase is reached without a very expensive repetition of the previous phases. Consequently, if a fatal problem is uncovered in the validation portion of the implementation phase of the phased life-cycle approach it is very expensive to correct.

The beauty of the phased life-cycle approach is its simplicity. The model is easy to understand, easy to represent, and allows for the definition of specific milestones even if they are hard to reach in actual practice.

Another software life-cycle model is the Prototype Model shown in Figure 2. The model emphasizes the source of product requests, the major go/no-go decision points, and the use of prototypes. A prototype is a model or a mockup of the software product. In contrast to a simulation model, the prototype model exhibits components of the actual product although normally at reduced capability or performance standards [Ref. 1:p. 49]. Prototyping allows designers to explore various technical issues and/or to allow the gradual development of requirements and performance specifications.

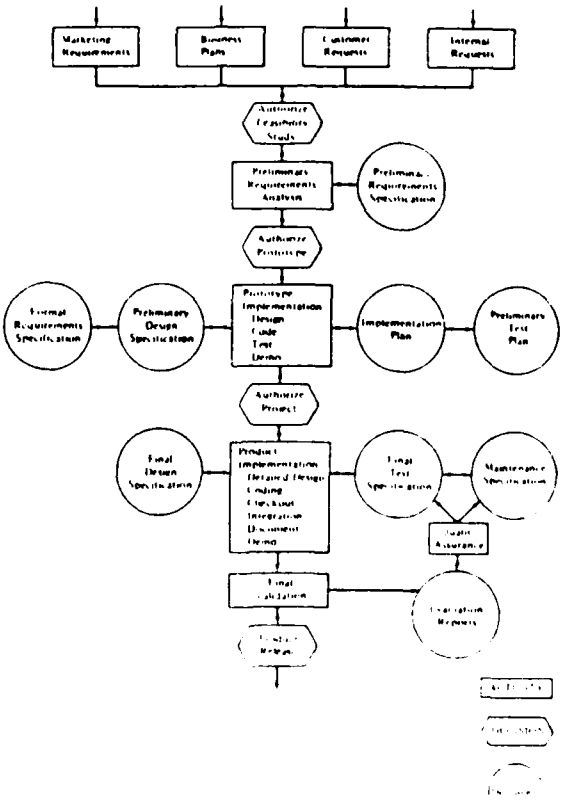


Figure 2: The Prototype Software Life-cycle Model [Ref. 1:p. 51]

The approach is useful when there is not a clean set of system requirements and the possibility of system requirements changing is high. Prototypes are designed to allow experimentation and change without the expense of full implementation, and to inexpensively identify the errors normally present in the first attempt to develop a system.

Critics of the prototyping method cite the "Let's go with the prototype" attitude that never develops the full system, or the expense involved with larger systems of building the system twice. However, when developing a new system from the beginning, some form of prototyping is usually desirable.

Yet another life-cycle model is the iterative enhancement model. In this model each version is a complete system that performs useful work. Enhancements are made to the previous system to add new capabilities as required. Some minor redesign of the previous system may occur to correct design deficiencies evident in the previous system; however, the majority of change is in the form of system enhancements.

As stated earlier, different models exist for different kinds of projects. Each emphasizes different aspects of the software life-cycle, and in many cases more than one type of model is combined to allow the development of a model specifically tailored to the project at hand. The most

important idea to be captured is the need for a life-cycle model. Such a model should encompass all the activities required to define, develop, test, deliver, operate, and maintain a software product. Many models recommend specific documents be produced at different phases to contain the documentation suggested for that phase. No single model is appropriate for all software products. However defining a model early on in the product development and identifying the planned documents to be produced is essential to the product's success.

The next section will view some recommended documents and the documentation contained in these documents as a function of the life-cycle phases.

B. DOCUMENTATION WITHIN THE SOFTWARE LIFE-CYCLE

Computer programs evolve in phases from the time that an idea to create or modify software occurs through the time that the software engineering process produces the required output. Using the terminology defined in the National Bureau of Standards Federal Information Processing Standards (NBS FIPS) publications [Ref. 3], the three major phases of the software project are: the initiation phase, the development phase, and the operation phase. The three phases, along with their associated sub-phases and suggested documentation documents are shown in Figure 3. This thesis is concerned

with the suqqested documentation collected during these phases.

INITIATION PHASE	DEVELOPMENT PHASE				OPERATION PHASE
	Definition Stage	Design Stage	Programming Stage	Test Stage	
PROJECT REQUEST DOCUMENT	Functional Requirements Document	System/ Subsystem Specification	Users Manual		
FEASIBILITY STUDY DOCUMENT		Program Specification	Operations Manual		
COST/BENEFIT ANALYSIS DOCUMENT	Data Requirements Document	Data Base Specification	Program Maintenance Manual		
		Test Plan		Test Analysis Report	

Figure 3. Documentation Within the Software Life-cycle [Ref. 3:p. 6]

Project Request Document. The purpose of this document is to provide the means for a user organization to request the development, procurement, or modification of software. It serves as the initiating document in the software life-cycle, and provides a basis for communication with the requesting organization to further analyze requirements and assess impacts. This document is quite often embedded in another document as a part of a larger system.

Feasibility Study Document. The purpose of the feasibility study document is to provide: (1) an analysis of objectives, requirements and concepts; (2) to evaluate alternative approaches; and (3) to identify the proposed solution and the justifying arguments that make this the most attractive alternative . This document, combined with the cost/benefit analysis document, should provide management with the required information to make an informed decision on whether or not to continue the project.

Cost/Benefit Analysis. The purpose of this document is to provide managers, users, designers, and auditors with adequate cost and benefit information to analyze different alternatives from the standpoint of the cost and benefit tradeoffs.

Functional Requirements Document. The purpose of this document is to provide a basis for a mutual understanding between all concerned parties about the results of the initial definition stage of the software, including the requirements, operating environment, and development plan.

Data Requirements Document. The purpose of this document is to provide, during the definition stage, a data description and technical information about data collection requirements.

System/Subsystem Specification. The purpose of this document is to specify for analysts and programmers the requirements, operating environment, design characteristics,

and program specifications (if desired) for a system/subsystem.

Program Specification. The purpose of this document is to specify for programmers the requirements, operating environment, and design characteristics of the computer program.

Data Base Specification. The purpose of this document is to specify the identification, logical characteristics, and physical characteristics of a particular database.

User's Manual. The purpose of this document is to sufficiently describe the functions performed by the software in a fashion that all users of the system might be able to understand (that is it uses non-ADP terminology).

Operations Manual. The purpose of this document is to provide computer operations personnel with a description of the software and of the operational evironment so that the software may be run properly.

Program Maintenance Manual. The purpose of this manual is to provide the maintenance programmer with the information necessary to understand the programs, their operating environment, and their maintenance procedures. This includes a listing of the code or instructions on how to obtain a listing.

Test Plan. The purpose of this document is to provide a plan for the testing of software; detailed specifications,

descriptions, and procedures for all tests; and test data reduction and evaluation criteria.

Test Analysis Report. The purpose of this document is to record and analyze test results and findings. Deficiencies and capabilities are presented for review and provide a basis to determine software readiness for implementation.

The next section will describe the reasoning behind recommending all these documents and the documentation contained within them.

C. TECHNICAL DOCUMENTATION GOALS

The purpose of good technical documentation, that documentation which deals with for example, program requirements, program design, interfaces, data requirements, algorithms, structures, etc., can be summarized in one phrase; to accommodate change at a reasonable cost. Furthermore the documents which contain this documentation provide definitive work products to be produced in each phase of the life-cycle. If a project were to meet all of its requirements and specifications during the testing phase, to maintain the same individuals throughout, and to face a completely static environment for its entire life-cycle, then documentation on how it performed its functions would be absolutely worthless after completion of the project except to the curious. However this is most always

never the case. The foreword to DOD-STD-1679A (Navy) [Ref. 4:p. iii] states three major factors in the design and documentation of Navy tactical programs. These are:

> Criticality of Performance. The combat capability of defense systems and the combat survivability of combatant units of the operating forces depend, in part, upon the effective operation of the software. Therefore, careful, persistent management must be exercised in the software development phase to ensure maximum reliability and maintainability.
>
> Changing Operational Requirement. Software implements system operations and doctrine in areas susceptible to many changes of performance requirements and specifications. These changes often impact the software and need expeditious implementation. This demands that software be designed to facilitate efficient change, sometimes at the expense of technical design efficiency. Designers must continously consider the tradeoffs between future modifiability of the product and design efficiency as the requirements now exist. Continuation of an efficient change capability over the operational life of the system also requires detailed documentation describing the system and the software. Proposed changes and their total impact must be easily discernible and must be capable of being implemented by personnel not associated with the original development effort.
>
> Life-cycle Cost. Development and implementation of changes to the software over the operational life of the system are costly. The design of the software during development must be strongly influenced by factors which will reduce life-cycle costs.

That the underlying goal of any documentation is to provide communication of the characteristics of a system and the processes used in developing the system independant of individuals, is apparent from the foreword to DOD-STD-1679A (Navy) and other publications. Technical documentation must anticipate change in the programs. Enough information, through flowcharts, listings, data dictionaries, etc., must

be available to persons not associated with the original design group to allow them to develop an understanding of what the program does and how it does it.

The next section will discuss the Navy's methods for providing this technical documentation as prescribed by DOD-STD-1679A(Navy) and interpreted by the DDG-51 Combat System Software Development Project.

III. NAVY TECHNICAL DOCUMENTATION

As is made apparent in the previous sections, excellent documentation is an important portion of any software project. This requirement holds true for Navy software projects as well where a major consideration of the project is to plan for change. This section presents a description of the navy tactical software life-cycle as described in DOD-STD-1679A(Navy) and presents the planned documentation in an example Navy software project by describing the planned documents to be produced in each phase of the life-cycle.

A. THE NAVY TACTICAL SOFTWARE LIFE-CYCLE

"The software challenge is to control the design process for a complex of operational computer programs so that the resulting products can be integrated into a reliable, maintainable, and survivable combat system fully responsive to the mission requirements" [Ref. 5]. This is the opening paragraph to the DDG-51 Computer Program Development Plan. It is like the challenge of any major software undertaking, with the exception that the possibility of further change, modifications, and enhancements is much greater, and that the software project, being a governmental project, will always be under close scrutiny. The Navy tactical software

life-cycle is very much like the phased life-cycle described earlier with stages very similiar to those described in NBS FIPS Publications 38 and 64. Using the Aegis Shipbuilding Program, DDG 51 Computer Program Development Plan, as an example of the typical tactical software development plan, one can see in Figure 4 the five phases described in the development plan.

- System Definition and Design Phase[1]. This is a predecessor phase in which the functional baseline is established in the A level specification, which is a formalization of the top level requirements of the system. And the first level of the allocated baseline is created and recorded in the element B1 specification, which is the breakdown of the A level specification into logical elements such as the radar element .

- Computer Program Definition and Design Phase. This phase encompasses the definition of the computer program performance requirements, the establishment of interface requirements, and the specification of the software top-level design. The performance requirements, documented in the Program Performance Specification (PPS), are the driving force for every subsequent phase of the computer program development process from design through testing and delivery.The requirements incorporated in this document, along with preliminary interface definitions and early top level software design considerations, are reviewed by the Navy at the Preliminary Design Review (PDR) which serves to present the PPS for approval as the preliminary allocated baseline for further development. Based on the approved PPS, the finalized interface requirements and the top-level program design are developed and documented, respectively, in the Interface Design Specification (IDS) and the Program Design Specification (PDS). The Critical Design Review

---

[1]The DDG-51 software project is a Naval Sea Systems Command project. However, the vast majority of the project is performed through a contract with RCA and various subcontractors. Some Navy projects are developed totally "in-house", however the normal procedure is to issue contracts for the actual software development.

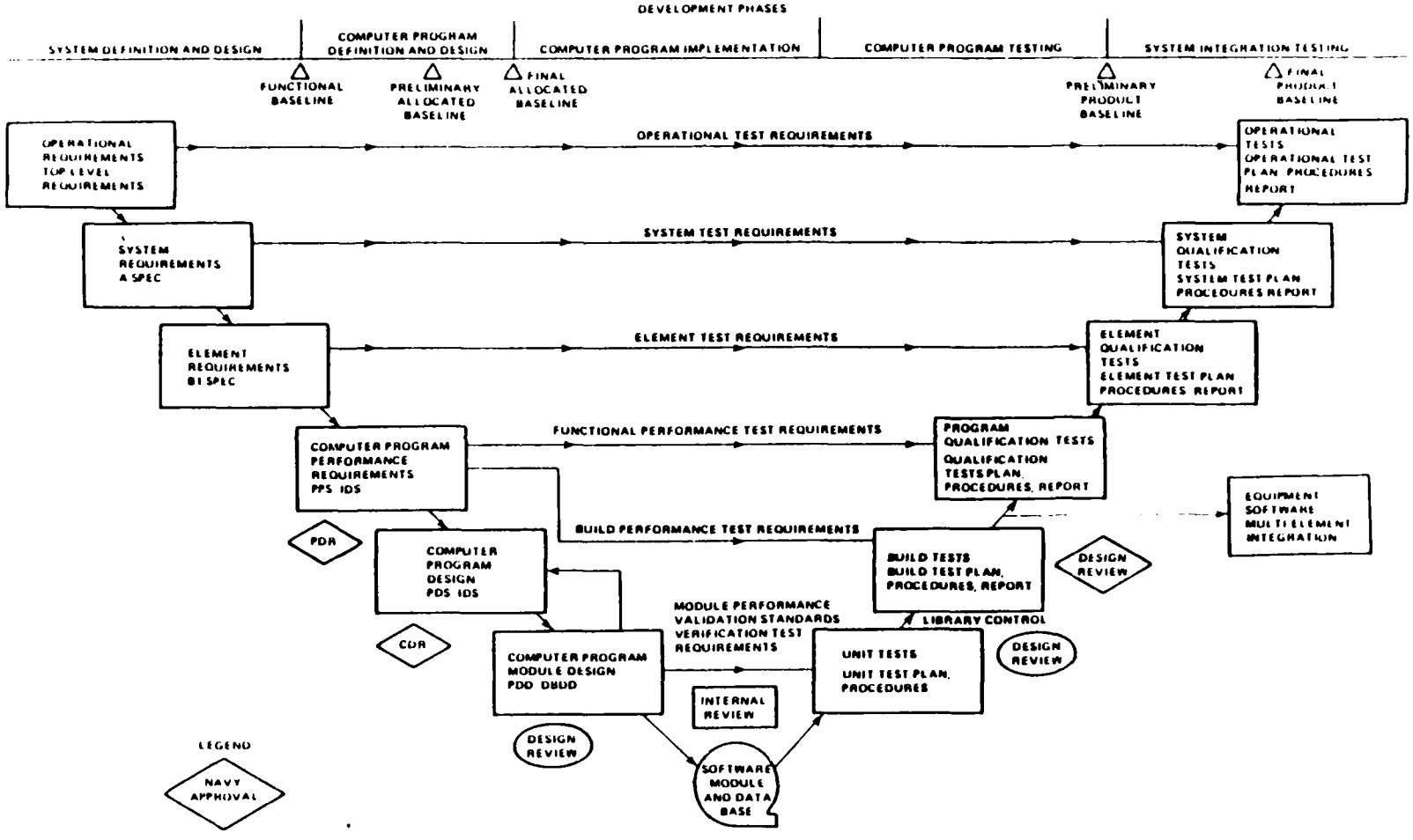


Figure 4. Naval Tactical Software Development Phases
[Ref. 5:p. 1-3]

(CDR) provides the mechanism for Navy review and approval of these documents. This completes the design phase of computer program development process, and these documents serve as the approved final allocated baseline for further development.

- Computer Program Implementation Phase. The computer program implementation phase is based on the approved documents and specifications produced in the design phase. The implementation phase encompasses the detailed design of the program modules and data base as well as the coding and debugging of these items. The program module logical designs and the detailed data base designs are developed and documented, respectively, in preliminary Program Description Documents (PDDs) and the Data Base Design Documents (DBD). These documents are reviewed at Informal Design Reviews, in which the Navy participates, and which serves to provide approval of the detailed design and authorization to proceed with coding. Once coding is completed and error free compiles of the modules and data base are achieved, an internal structured walk-through of the implemented code is undertaken to assure compliance with design requirements. Successful completion of this structured walk-through serves to release the modules and the data base for testing. This completes the implementation phase.

- Computer Program Testing Phase. Computer Program development te sting is performed within the context of the top-down approach to development. Testing starts with the smallest operating components; i.e., modules, and develops through successively more complex and inclusive stages. Modules are integrated into subprogram builds, which are operational subsets of the complete computer program. Build tests are performed with Navy participation. Functional capabilities are added to the subprogram builds, and, in the last stage, the final build is tested as a complete computer program. Test plans, test procedures, and test reports are prepared at all levels of testing, beginning with the module unit testing. The Computer Program Qualification Test, conducted at the developer computer program test facility and performed to Navy approved test procedures, is the final test of the computer program as a standalone entity. The successful accomplishment of this test marks the completion of the software development phase. Subsequent activity is in support of element and system level integration and testing. A preliminary product baseline is established at the completion of the software testing phase.

- System Integration Testing Phase. At the Combat System Engineering Development (CSED) site, software is employed as embedded subsystems for integration with equipment and/or other computer programs in the element and multi-element testing environment of that facility. Essentially conditions as close to the actual operational environment of the final product as possible are constructed for complete system testing to include an actual mockup of the platform the system will be embedded in. The final product baseline is established at the completion of system qualification testing.

	INITIATION PHASE	DEVELOPMENT PHASE				OPERATION PHASE
NBC PIES		DEFINITION STAGE	DESIGN STAGE	PROGRAMMING STAGE	TEST STAGE	
	- PROJECT REQUEST DOCUMENT -FEASIBIL. STUDY DOCUMENT - COST/ BENEFIT ANALYSIS DOCUMENT	- FUNCTIONAL REQUIREMENTS DOCUMENT - DATA REQUIREMENTS DOCUMENT	- SYSTEM/ SUBSYSTEM SPECIFICATION - PROGRAM SPECIFICATION - DATABASE SPECIFICATION	- USER's MANUAL - OPERATIONS MANUAL - PROGRAM MAINTENANCE MANUAL	- TEST ANALYSIS REPORT	
			-TEST PLAN-			
[illegible]-STD-[illegible]A	- SOFTWARE DEVELOPMENT PLAN - QUALITY ASSURANCE PLAN - CONFIGURATION MANAGEMENT PLAN	- PROGRAM PERFORMANCE SPECIFICATION - DATABASE DESIGN DOCUMENT - PROGRAM DESIGN SPECIFICATION - BUILD PLAN AND INTERFACE DES. SPECIFICATION - COMPUTER	- PROGRAM DESCRIPTION DOCUMENT - PROGRAM PACKAGE DOCUMENT - SYSTEM OPERATOR'S MANUAL - MODULE DEVELOPMENT FOLDER	- PROGRAM TEST PROCEDURES - PROGRAM TEST PLAN - PROGRAM TEST REPORT		SOFTWARE TROUBLE REPORT SOFTWARE CHANGE PROPOSAL
			PROGRAM TEST SPECIFICATION-			
	SYSTEM DEFINITION AND DESIGN PHASE	COMPUTER PROGRAM DEFINITION AND DESIGN PHASE	COMPUTER PROGRAM IMPLEMENTATION PHASE	COMPUTER PROGRAM AND SYSTEM INTEG. TESTING PHASE		OPERATION PHASE

Figure 5. Comparison of Life-cycle Documentation Requirements

Figure 5 shows the similarities between the Navy's phased life-cycle and the NBS phased life-cycle by displaying the five main phases of naval tactical software and the phases recommended by the National Bureau of Standards. Also shown is the suggested documents to be produced during these phases.

The concept of documentation requirements for the Navy is contained in DOD-STD-1679A, however the actual document requirements, and the specific contents of each document, along with the prescribed layout of the document is prescribed in a data item description (DID). These DIDs are envoked in a contract as necessary, to ensure standardized and complete documentation is included in the software package and to provide the flexibility to tailor DOD-STD-1679A(Navy) to various software projects. The documentation requirements and the documents to be produced in Naval tactical software development is discussed in the next section.

B. NAVAL SOFTWARE TECHNICAL DOCUMENTATION REQUIREMENTS

As stated before, the content, style, and coverage of all documentation associated with a naval software contract is well defined in the contract. The "Contract Data Requirements List", or CDRL, specifies which military standard the contract shall conform to. The military standard, MIL-STD-1679A (Navy) in this case, is not simply a

quideline, it is much stronqer. A standard must be complied with, and qenerally provides the "shalls" of a contract, or the qeneral areas that must be addressed throuqh the documentation. Also included in the CDRL are "data item descriptions", or DIDs, which specify exactly how a document will appear, and exactly what this document will contain. The standard provides the requirements in qeneral terms, and the DIDs provide the specifics for a particular contract. DIDs provide the flexibility to tailor a military software contract to any project, be it larqe or small.

The best way to understand the documentation required in a military software project is to view it as a it is to be recorded durinq the life-cycle.

- Initial Planninq Software Development Phase:
  - Objectives:
    1. To combine the Navy's and contractor's ideas for accomplishinq the project.
  - Documentation:
    - Software Development Plan. Software manaqement's plan for developinq the proqram performance specifications, and producinq the software.
    - Software Configuration Manaqement Plan. The configuration manaqement qroup's plan for managinq chanqes in software configuration durinq software development.

- Software Quality Assurance (QA) Plan. The QA group's plan for verifying that all requirements in the contract are met. The basis for the test plan.

- Computer Program Definition Phase:
  - Objectives:
    1. To identify the Computer Program Configuration Items (CPCIs) required for each element.
    2. To determine the detailed program performance requirements for each combat system element computer program and to specify them in the element Program Performance Specification (PPS).
    3. To define the interface design requirements and to specify them in the Interface Design Specification (IDS).
    4. To define the operating system and support programs required to support the operational element programs and the development process.
    5. To provide design information to the Navy, system designers, and other engineering agents.
    6. To reduce risk by establishing the technical feasibility of the Combat System Software.
    7. To identify critical areas early in the

software development cycle.

8. To provide for quality assurance and testing requirements.

- Process:

This functional baseline established during the system definition and design phase includes the element Prime Item Development specification (B1) which provides the vehicle for mapping the functions allocated to computer programs into computer program performance requirements. Figure 6 shows the process used in determining these computer program performance requirements. The resulting definition of element computer program requirements is documented in the PPS for each element, the preliminary Interface Design Specification (IDS), and the Design Disclosure Package (DDP). These documents form the basis for the Preliminary Design Review (PDR).

- Documentation:

  - Program Performance Specification. For each element the PPS provides the baseline document for subsequent computer program development. It defines the operational and functional performance required of the element computer program, and provisions for quality assurance and testing. The PPS also specifies a computer

equipment configuration designed to satisfy the specified requirements.

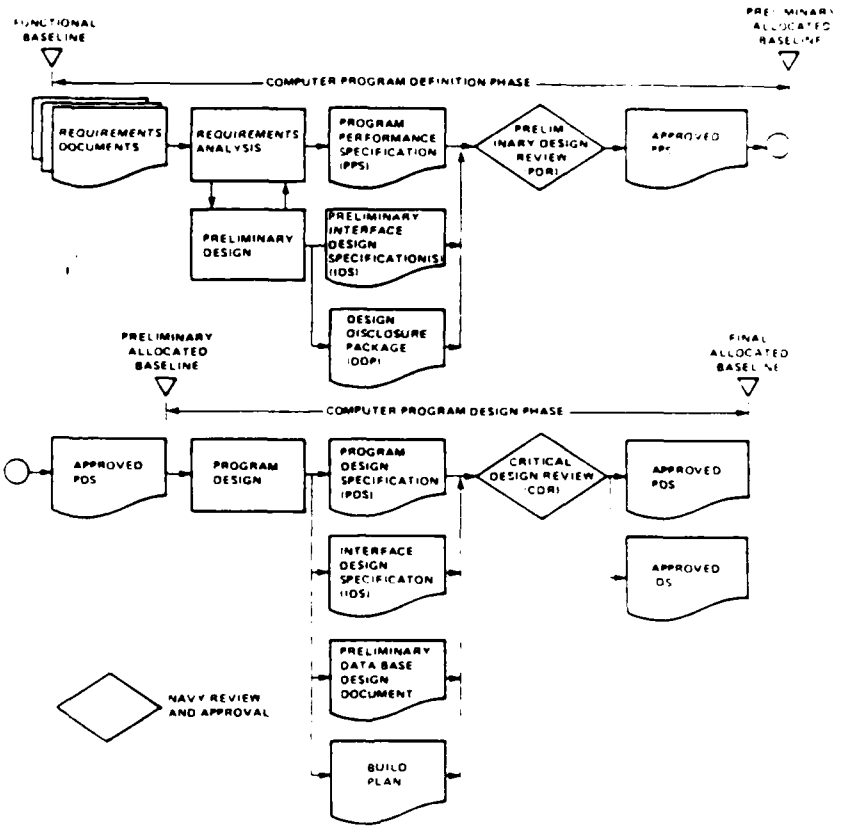


Figure 6. Computer Program Definition and Design Phases [Ref. 5:p. 3-4]

- Interface Design Specification(s). The preliminary IDS provided the definition of all digital interfaces to the element computer program.
- Design Disclosure Package. The DDP provides the results of modeling, system engineering analysis,

and any other studies done to determine system feasibility.

- Computer Program Design Phase
  - Objectives:
    1. To develop the computer program architecture and top-level design and to specify it in the Program Design Specification (PDS) for each element.
    2. To specify module functional descriptions, program control logic, high-level data base design, and initial memory and time resource budget requirements.
    3. To develop detailed definition of computer program interfaces, and to specify these in computer program interface design specifications (IDSs).
  - Process:

    Following the approval of the PPS for an element computer program, an element Program Design Specification (PDS) is developed to specify the architectural and top-level design requirements for the element computer program. The computer program design process involves the allocation of software functions defined in the PPS to software tasks, as shown in figure 7. Some of the functions to be performed during this phase are:

    - A functional allocation of all performance

requirements shall be made to the computer program modules.

- A functional flow of program data and control shall be defined in all modes of operation.
- The proposed architecture shall be verified as to its capabilities to support the maximum computational load.
- A common data base shall be designed for all data elements used by two or more subprograms.

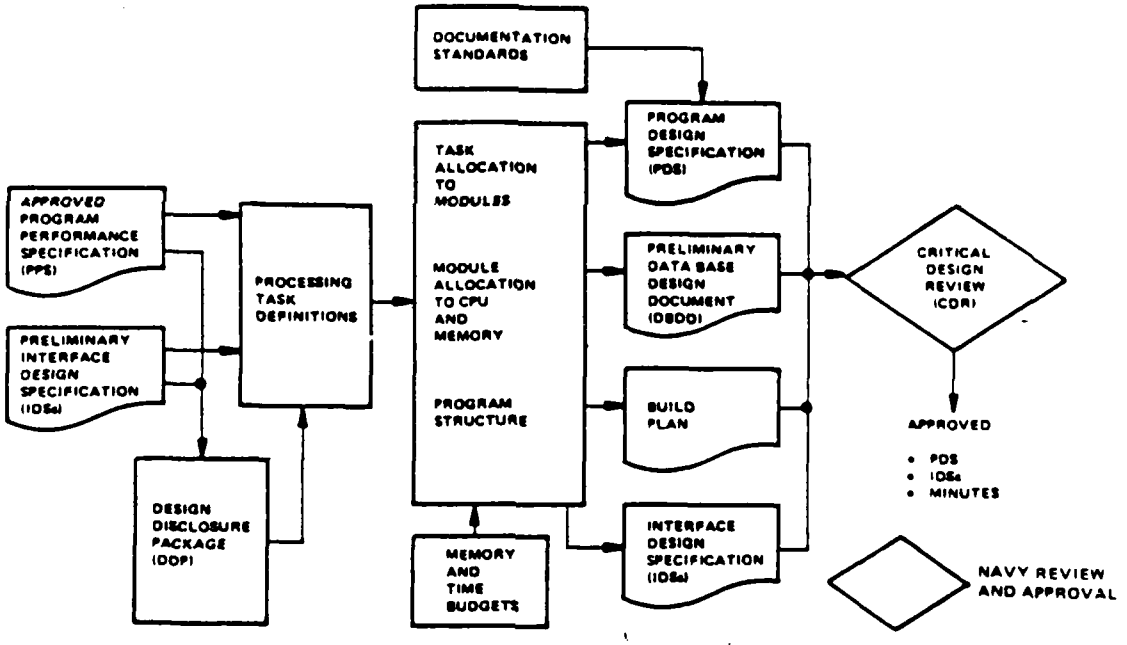


Figure 7. Computer Program Design Process [Ref. 5:p. 3-6]

- Documentation:
  - Program Design Specification. The PDS is generated according to the requirements and constraints laid down by the PPS. At this stage, the development process focuses on a top-down translation of system operational function requirements into program logic including module functional descriptions, program control logic, and memory-and-timing estimates. Such items are necessary for detailed subprogram and data-base design and implementation. Included are program functional flow diagrams, cross reference tables between the PPS and the PDS, the modular program structure, and the program data flow diagram.
  - Interface Design Specification. The preliminary IDS(s) produced during the definition phase are updated, and the details of interfaces (messages) are added. This document, after approval at the Critical Design Review, is placed under configuration control. This document provides a detailed description of: all data units, all messages, and all control signals.
  - Data Base Design Document. A DBD is produced for each element. The DBD provides a complete detailed description of all common data items necessary to

carry out the element computer program functions. Common data are those required by two or more modules. This document is completed during the next phase, i.e., the implementation phase. During the detailed design portion of the implementation phase, an element computer program data base librarian maintains the the developing data base in the form of the DBD. Using the DBD in the role of Configuration Management, the DBD serves the purpose of (1) Controlling the data elements definitions, (2) Maintaining the attributes of fields, (3) Reducing the data redundancies and inconsistencies, (4) Allowing module designers to communicate effectively with each other through joint meetings prior to changing the data base, (5) Containing cross references of users of data, and (6) Determining the impact of data base changes on the data base and other modules.

- Build Plan. A "build" is a logical collection of modules. Modules are designated as part of a build, tested as modules, then integrated into builds. Build 1 may have modules 1,3,5,8. Build 2 is modules 1,2,3,5,8. Build 2 is not dependant on completion of build 1; however a certain level of completion must be achieved prior to

the modules being used in build 2. Each build adds modules with phased integration until the complete product is achieved. The build plan is the design of this process, and is

Phase	Activities	Pre-Review Products*	Reviews	Post-Review Products*
Computer Program Definition	Perform requirements analysis. Prepare PPS	Program Performance Specification	Preliminary Design Review (PDR)	Approved PPS
	Prepare preliminary IDS(s)	Preliminary Interface Design Specification(s) (IDS)		
	Conduct preliminary software design	Preliminary Design Review (PDR) Presentation Material including Design Disclosure Package (DDP)		Approved PDR Minutes
Computer Program Design	Conduct computer program design. Prepare PDS	Program Design Specification (PDS)	Critical Design Review (CDR)	Approved PDS
	Prepare IDSs	IDS(s) Build Plan		Approved IDS(s)
	Conduct build planning	Build Plan Preliminary Data Base Design Document (DBD)		
	Conduct data base design	Critical Design Review (CDR) Presentation Material		Approved CDR Minutes

*Audited and maintained by Quality Assurance Organization for the duration of project.

Figure 8. Summary of Computer Program Design Approach [Ref. 5:p. 3-9]

maintained by the manager responsible for the software development and is updated as required.

- Computer Program Implementation Phase

The implementation phase is actually made up of two sub-phases; the detailed design phase, and the code and debug phase.

- Detailed Design Phase:
  - Objectives: After successful review of the high-level module designs, the detailed design of each module is begun. The data design, tables, variables, flags, indicies, data base references, I/O formats, required system library routines, conditions for initiation, module limitations, and interface descriptions are defined. Each of the module requirements identified in the PDS must be designed into the module.
  - Documentation:
    - Program Description Document (PDD). Provides a complete technical description of all module functions, structures, operational environments, operating constraints, and private data base organization, for each module of the element computer program. Each module is described in its own volume of the PDD with referenced appendixes as computer printout listings. The PDD describes

and completely defines the basic logic and program procedures for each application module. As a detailed description of the module structure, the PDD serves as the essential instrument for subsequent use by operational maintenance and contractor personnel diagnosing troubles, making adaption changes, designing and implementing modifications to the system, and in adding new functions to the completed program.

- Data Base Design Document. Described earlier.
- Module Development Folder (MDF). The MDF for each module is begun after the internal design review, and is maintained by the cognizant programmer during all phases of the module development and test. Items contained in the MDF include: (1) Results of the internal design review, (2) Evidence of approval to proceed to the next phase, (3) Resolution of action items, (4) Data describing the rational for the module design, (5) The indented source listing generated by ASCP, (6) Results of the structured walk-through, (7) Description of the module for the PDD, (8) Unit test plan and procedure, and (9) Unit test results.

- Code and Debug Phase.

After the detailed design is completed and

approved, the code and debug of the module begins as shown in Figure 9. The programmer/analyst(s) generates source code that satisfies the detailed design of the module. Module and data base coding is considered complete when a clean, error free compile is achieved and the Automated Source Code Processor (ASCP) computer program has verified adherence to structured programming standards and conventions. Other than updating the MDF, there are no additional documents produced during this portion of the implementation phase.

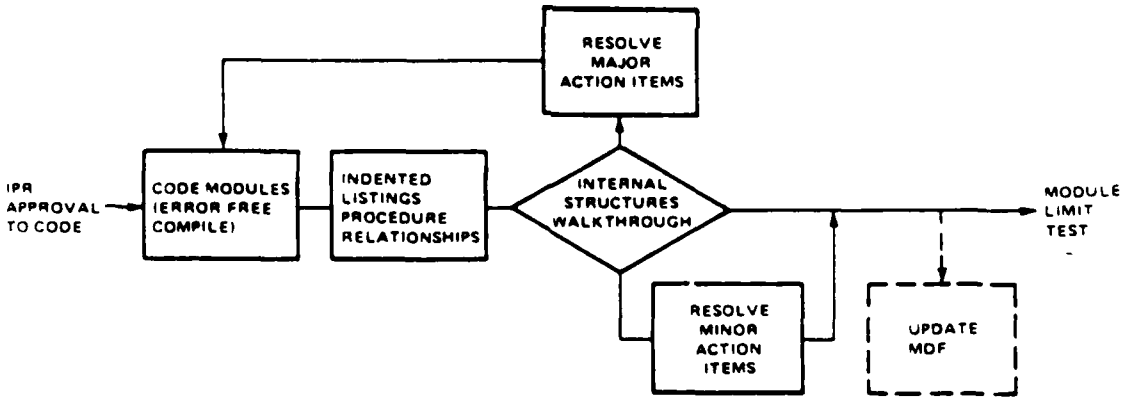


Figure 9. Code and Debug Process [Ref. 5:p. 4-8]

Phase	Activities	Pre-review Products	Reviews	Post Review Products*
Detail Design	High Level module design	High-level flow diagrams	Design Review of detailed design	Approved PDD (Annotated)
	Detailed module design Prepare PDD Data design Prepare DBD Develop Module Development Folder (MDF)	Module Development Folder (MDF) Program Description Document (PDD) Data Base Design Document (DBD)		Approved DBD (Annotated)
Code and Debug	Develop source code Eliminate source code errors Produce indented listings and flow procedure relationships Update MDF	Coded modules PDD (Annotated) DBD (Annotated) Updated MDF	Internal structured walk-through	Debugged code

*Quality Assurance control for the duration of project

Figure 10. Summary of Detailed Design and Code Approach [Ref. 5:p. 4-9]

- Computer Program Testing Phase.
  - Objectives: This phase actually spans the implementation phase and its own phase. Testing is divided into three distinct subphases, the unit test, the build test, and the program qualification test. Testing starts with the smallest operating component, and develops through successively more complex and inclusive stages. The modules are integrated into subprogram builds which are operational subsets of the complete program. Build level tests are performed

on these integrated builds to validate functional capabilities. Additional functional capabilities are successively added to the subprogram builds and, in the last stage, the final build is tested as a complete program. Naval participation in all aspects of the testing phase is required. Additionally Internal Program Reviews are held to gain joint Navy, contractor, and subcontractor agreement with test definitions, content, methodology, performance, and evaluation for build and qualification tests.

- Unit Test Plan: The unit test plan is contained in the MDF. It contains the strategy to exercise all of the code in the module, either in a standalone environment, or in an integrated environment with previously tested modules. Any failure is documented in an action item, which is included in the MDF and returned to the programmer for correction. Also a part of the unit test plan is the unit test procedure. Unit test procedures are derived from the unit test plan, and corresponding design documentation. They present detailed instructions for test setup, execution, and evaluation of test results. The unit test procedure also becomes part of the MDF.
- Build Test Plan and Procedure: The build test plan and procedure document contains the following information:

1. A definition of the testing program and strategy required to test the build, including a rational for for the testing program as it relates to the functional capabilities and structure of the build.
2. An outline of the capabilities of the build to be tested, plus those capabilities provided, but not tested, and capabilities previously tested that require retest.
3. A description of the test methods, test tools, and observations and measurement techniques to be used.
4. A specification of the test sequence.
5. A description of the contents of the test, including personnel requirements, responsibilities, and facilities required.
6. Detailed instructions for test setup, execution, and evaluation of test results.
7. A description of the scenario(s) which demonstrate the major operational capabilities of the build.

- Program Qualification Testing. Program qualification testing is performed at the Computer Program Test Site and is the final stage of the computer program development phase. Here the build test plans and procedures come together with the PPS, the IDS, the preliminary user's manual, and the preliminary operators manual to generate the Program Qualification Test Plan and Procedures. Upon completion of the

testing the Program Package and the Operator's Manual are delivered for operational use. Products of this phase are the Program Qualification Test Report and the Test Discrepancy Report.

Figure 11 summarizes the relationships between the required software documents and software phases.

SOFTWARE DOCUMENTS	INITIAL PLANNING PHASE	REQUIREMENTS ANALYSIS PHASE	PRELIMINARY DESIGN PHASE	DETAILED DESIGN PHASE	CODE, DEBUG AND UNIT TEST PHASE	PROGRAMMING TEAM TESTING PHASE	TEST TEAM TESTING PHASE
SOFTWARE DEVELOPMENT PLAN (SDP)	p, f						
SOFTWARE CONFIGURATION MANAGEMENT PLAN (CM PLAN)	p, f						
SOFTWARE QUALITY ASSURANCE PLAN (Q/A PLAN)	p, f						
SOFTWARE STANDARDS AND CONVENTIONS		p					
PROGRAM PERFORMANCE SPECIFICATION (PPS)		p, f	u	u			u
PRELIMINARY DESIGN SPECIFICATION (PDS)			p, f	u			u
PROGRAM DESCRIPTION DOCUMENT (PDD)				p, f			u
DATA BASE DESIGN DOCUMENT (DBDD)				p, f			u
TEST PLAN		p, f					
TEST SPECIFICATION(S)			p, f			u	u
TEST PROCEDURES				p, f		u	u
ACCEPTANCE TEST PLAN							
OPERATOR'S MANUAL					d		p, f
SYSTEM OPERATOR'S MANUAL					d		p, f

PRELIMINARY DESIGN REVIEW (PDR)

CRITICAL DESIGN REVIEW (CDR)

LEGEND
- p PRELIMINARY
- f FINAL
- u UPDATED
- d DRAFT

Figure 11. Relationships Between Software Documents and Software Phases. [Ref. 6:p 402]

IV. NAVAL DOCUMENTATION: PROBLEMS AND SOLUTIONS

A. NAVAL DOCUMENTATION, HOW DOES IT STACK UP?

When viewing the Navy Military Standard for Software Development, DOD-STD-1679A(Navy), one cannot help but to be impressed with the very modern and complete approach the Navy takes in its software development. Using the DDG-51 software design effort now in progress, and the CG-41 design effort as an example of a major tactical embedded computer software design program, every facet of modern software design technology is present. Software conforming to DOD-STD-1679A(Navy) exhibits the following characteristics:

- A well defined software methodology that includes a defined life-cycle model and definitive phases of the life-cycle.
- A strong level of planning in the System Definition and Design Phase which is exceptionally well documented through the System Requirements (A Spec), and Element Requirements (B1 Spec).
- Early definition of the Computer Program Performance Requirements documented in the PPS and early definition of interfaces documented in the IDS.
- Modular computer program design specified in the PDS with additional updates to the IDS and further documented in the PDD and DBDD.
- A strong formal configuration management plan that is enacted early in the design phase to maintain a definitive version of the project through all phases, and managed by a separate configuration management group.
- Early definition of performance requirements directly translated into test plans and procedures.

- A highly structured test plan that includes test requirements, test procedures, definitive metrics for the test included in the test specification, test documentation procedures for conducting the test, procedures for recording the results and gaining approval, and a cyclic approach to correction of errors detected.

- A quality assurance program that incorporates all aspects of the testing, with a seperate QA team.

- A software methodology that is designed to smoothly produce the documents required as a function of following the prescribed methodology.

- Standardized document content and design as specified in data item descriptions (DIDs) enacted in the contract.

- A well controlled review procedure as part of the methodology that ensures the "right product is being built", and that the "product is being built right", and ensures that the Navy is well informed and in concurrance with the contractor in all phases of the project.

Additionally, the software development methodology utilized, and the documentation produced under DOD-STD-1679A(Navy) meets or exceeds the requirements/recommendations of the following publications:

- National Bureau of Standards (NBS) Federal Information Processing Standards (FIPS) Publications 38 and 64, Guidelines for Documentation of Computer Programs and Automated Data Systems, Guidelines for Documentation of Computer Programs and Automated Data Systems for the Initiation Phase, respectively.

- National Bureau of Standards, Special Publication 500 -15, Documentation of Computer Programs and Automated Systems. A symposium held at NBS in 1976 to discuss the problems in documentation of computer programs. All problems addressed in this symposium and listed in the publication are addressed in DOD-STD-1679A(Navy).

- Institute of Electrical and Electronics Engineers (IEEE) Standard for Software Test Documentation, IEEE Std 829 -1983. Although the terminology differs somewhat, the major provisions of the IEEE standard are covered well

in DOD-STD-1679A(Navy).

- National Bureau of Standards Special Publication 500 -106, Guidance on Software Maintenance. An important note here is that the Navy's definition of maintenance and that of the NBS differ. Whereas the NBS terms any changes to the code after the approval of the original baseline as maintenance (termed perfective and adaptive maintenance), the Navy considers modifications and additions of enhancments to the original baseline as further development. However, the provisions of NBS Special Publication 500-106 for perfective and adaptive maintenance are included in DOD-STD-1679A(Navy).

B. NAVAL DOCUMENTATION, THE PROBLEM AND CAUSES

Although software developed and documented in accordance with DOD-STD-1679A(Navy) and the associated envoked DIDs would appear to utilize what is presently considered to be the most modern and effective software development methodologies, and to provide documentation that meets the provisions and requirements of all authoritative publications there are still major problems expressed by personnel associated with naval software development. However, the cause of the problems do not appear to be as a result of discrepancies in the applicable standards, but rather the problems appear to be as a result of management improperly viewing the reasons for documentation and not placing the appropriate priority on the documentation process.

One problem noted is the possibility of "over-documenting" a project. Documentation is an expensive undertaking, and it is money "up front", that is money that

is required early in the project's life-cycle. Additionally, modifications to the software made in the later phases require that the project go back and update all of the documentation affected by the change. This is always a major expense. The more complex the level of the affected documentation then the more expensive the modifications become. Dr. Singh of Naval Material Command 08Y expressed in an interview a strong belief that many of the smaller tactical software projects are over documented, or that the documentation level is much too complex for the scope of the project. He proposes that one of the first actions of the project management is to determine exactly what documentation level is desired based on the criticality of the project, the size of the project, its planned life-span, and the probability of changes being made in the original requirements specification. His view is supported in FIPS PUB 38 which defines four levels of documentation [Ref. 4: pp. 10-11]. However Dr. Singh feels that even if the software is "critical", which would place it in the highest level of FIPS PUB 38 documentation, there is some room for making the documentation a bit less complex. While too little documentation normally brings about major expense later on in the life-cycle, too much documentation acts as an unnecessary burden and expense throughout the life-cycle.

Another problem evolves from the Navy's commitment to the phased life-cycle approach. The phased life-cycle

approach does not provide for changes being made in products of phases already past such as the requirements specification being changed when the project is in the testing phase. If a change occurs then the only way the change can be accommodated is to back up to the last phase that would not be affected by the change and start over with the development process at that point incorporating the change as the project flows through the phases. This is an expensive and time consuming method to accommodate change but it is the only method that will ensure success when using the phased life-cycle approach.

The most significant problem expressed concerning documentation of naval tactical software projects was a lack of understanding of what purpose the documentation is to serve. Documentation serves two major purposes; (1) to interact with the software methodology used in providing a guide for accomplishment of the project, a set of wickets if you will, for the engineers to pass through on their way to accomplishing their goals, and (2) to historically record the goals or requirements of the system, and the processes and methods used to achieve those requirements as a basis for understanding the system in order to be able to modify the system in the future. DOD-STD-1679A(Navy) provides a fairly good methodology outline for software design of a naval tactical system. It also provides an excellent description of what, historically, have proven to be useful

documents to both the designer and the maintainer. What it does not provide is an understanding of how important documentation is to the software development effort or how important documentation is to the continual success of the project in the face of change. Documentation is an integral part of planning and controlling the software development. Each document represents a milestone in the further reduction of top level requirements into accomplishable tasks. It is the tangible portion of the methodology that functions as a control tool for management throughout the life-cycle of the software. And it allows the product to be enhanced or otherwise modified in the future by individuals not originally associated with the development effort. Management must realize this as such and enforce the discipline necessary to produce or update the documentation.

Many of the individuals interviewed for this thesis expressed a concern that the documentation for their project was either non-existant, very late, or not up to date with the actual state of the project. Documentation that is not up to date is in many cases worse than non-existant documentation in that it has the possibility of misrepresenting the system. Documentation that is late normally turns into documentation that is not up to date. In most software design methodologies documentation is used to provide a measure of what the system goals are, and where the system is. If there is nothing that provides a

definitive answer of where the system is, then it is quite difficult to say where the system is going.

Although an excellent outline of what documentation to consider is provided in DOD-STD-1679A(Navy) and most projects commence with an excellent plan for documenting their systems, the latter stages of a large portion of these projects find themselves with documenatation that is inaccurate or behind schedule. NBS Special Publication 500-87 [Ref. 7] provides five main reasons for this happening:

- Low Priority for Documentation. Project management does not encourage the system analysts and designers to maintain and update their documentation when faced with time schedules and limited resources.

- Lack of Resources. Low priority for documentation often leads to inadequate resources to perform necessary documentation tasks. The "we'll get to it when we have time and money" syndrome takes effect.

- Lack of Planning. Management fails to clarify documentation requirements at the start of the project.

- Failure to Specify. Management fails to adequately specify the system requirements at the start of the project.

- Personnel Attitudes. Programmers often have little interest in documenting. The work is often viewed as unglamorous, and daily pressures often override some perceived uncertain needs for documentation. Documentation is not visable; as long as the project can move along then documentation is viewed as unneeded and priorities often shift to more visable objectives.

C. DOCUMENTATION, A SOLUTION IN ITSELF

Documentation must not be viewed as a necessary evil in a software project, but rather as a vital management tool to

be used in controlling the entire project. Management must realize that documentation is a major key to a successful project. "In order to yield a good software product, the software documentation activities must be integrated into the whole software development process. Program documentation is an active part of program development. It should not be treated as a passive task of simply recapturing the descriptions of an already developed program. Good program design leads to good documentation. Good documentation contributes to good design." [Ref. 8]

Viewing the causes of poor documentation mentioned in section B above, naval tactical software projects appear to suffer from only three of the five causes. There is neither a lack of planning nor a failure to specify initial requirements. In fact, naval tactical software projects place a high priority on initial planning and specifying the initial requirements of the project. Unfortunately, despite a seemingly high priority being placed on documentation as evidenced by the strong discussion of documentation in DOD-STD-1679A(Navy) and the associated DIDs, documentation appears to be quickly placed on the "back burner" when confronted with deadline dates and unplanned modifications. These actions betray a relatively low priority being placed on documentation. Additionally, when faced with funding limitations and unplanned modifications, updating documentation is again assigned a low priority, thus

becoming late or incomplete which complicates the project later in its life-cycle when good documentation is required for additional modifications or improvements. Personnal attitudes degrading the quality of naval tactical software documentation more difficult to substantiate through naval sources. The Navy normally contracts out the vast majority of its programming and design with a substantial approval and testing process to maintain control of the project. However a significant number of studies have concluded that programmers have neither the desire nor the exact talents to properly document a project [Ref. 9], so it can be safely assumed that naval contractors suffer from the same problems.

The solution to these problems must take on a two pronged approach. One must be the responsibilty of the contracted corporations, the other a responsibility of naval tactical software management.

There is presently quite a large amount of research and development being peformed in the areas of automated tools for the documentation effort. With the ever increasing power of computers, and the simultaneous demand for systems to take advantage of the developments in computers and to do more, the complexity of the major systems being produced today has outstripped the ability of the system analysts without the assistance of automated tools. The time has passed when a designer can effectively review module

performance requirements and say with certainty that they meet the top level requirements. There is simply too much for an individual to handle. Additionally, maintaining data dictionaries, data flow/control flow diagrams, interface specifications, and the like, has again become much too complex for the unassisted individual. Although beyond the scope of this thesis, it strongly recommended that naval managers, when evaluating a contractor's response to request for proposals, take into account the contractor's ability to produce the proper documentation completely and on time and their ability to maintain this documentation in the event of unplanned changes. Managers should consider what tools are being employed by the contractor and what, if any, documentation organization is proposed by the contractor.

The second half of the two pronged approach is the responsibility of naval tactical software management. First and foremost management must realize the importance of quality, timely documentation to the projects success. Documentation must be moved from the back to the front burner. This can be done by simply making documentation a factor in which the quality of the contract is judged. That is, subject documentation delivered to a metrics evaluation, the performance of which determines a portion of payment on the contract. A contractor faced with a loss of revenue, or a gain, as determined by the quality of his documentation

performance will certainly raise documentation in his priorities.

Additionally, management should consider the formation of a Documentation Group on a management level par with that of the Quality Assurance, and Configuration Management groups. Although the idea of a documentation committee is not new, a documentation group would be formed at the beginning of the project and relieve QA, CM, and designers/analysts of the burden of documentation decisions, not formed to review the already present problems in documentation as most committees are. A documentation group could be charged with the responsibility to:

- Recommend required documentation and document complexity for the project.
- Evaluate a contractor's ability to produce required documentation.
- Establish metrics with which the contractor's documentation performance could be judged.
- Perform auditing functions to verify documentation accurately reflects the system being produced.
- Work with QA and CM in maintaining documentation.
- Collect cost versus benefits data on documentation to analyze for use in future projects.
- Provide a group of individuals whose major concern is that of proper documentation of systems and maintenance of that documentation.

Whether a formal documentation group is established or not, the importance of high quality timely documentation as a critical portion of a successful project must be impressed

upon the entire development team. The responsibilities recommended above for the proposed documentation group must be fulfilled no matter what the organizational distribution is. Documentation must be maintained throughout the entire life-cycle and must receive a priority equal to that of the design and code function itself. If a documentation requirement is delayed in order to be able to meet some unforseen requirement, then management must make every effort to ensure that the documentation is completed as soon as feasible and not allowed to be continually delayed. The problems caused as a result of late documentation increase proportionately with the amount of delay involved. Additionally a direct correlation between the size and complexity of the proposed system and that of the documentation must be established, the amount of documentation and the complexity must be critically reviewed at the start of a project and recorded in the contract requirements.

The old adage "the job isn't done until the paper work is completed" is a most important rule of thumb to remember when managing a software development project.

V. SUMMARY OF CONCLUSIONS

This thesis has reviewed the methodology used in the development of Navy tactical embedded computer software and the documentation produced by following this methodology. The major conclusions are presented in the following paragraphs.

The methodology utilized by the Navy was compared to those recommended by the National Bureau of Standards, the IEEE, commercial publications, academic publications, and experienced individuals. Comparison revealed that the Navy utilizes an extremely modern, complete, and efficient methodology that incorporates most all of the suggested development procedures.

The Navy, in developing a new combat system, or in modifying an existing system, normally acts as a management team that contracts out explicit software design to a contractor through competitive bids. DOD-STD-1679A(Navy) acts as the major controlling document for Navy management to use in contractor developed software. This standard defines a general methodology using the phased life-cycle approach for software development that can be tailored to the specific project through the use of data item descriptions(DIDs). The standard does not address specific

development procedures such as Chief Programmer Teams, or technical writers, but rather it defines the output desired by the Navy and characteristics this output must exhibit for the product to be satisfactory. Items the product must exhibit include top-down design, modularity, bottom-up testing, etc.. The major controlling functions utilized in this standard are a strong interaction between contractor and the Navy, specific divisions where Navy approval is required for further development, and extremely specific documentation requirements that would normally be fulfilled if the contractor were following the provisions dictated by DOD-STD-1679A(Navy).

The DOD-STD-1679A(Navy) was found to be entirely satisfactory for the purpose to which it was designed when exercised by competent management.

The major problems discussed were not caused by deficiencies in DOD-STD-1679A, but rather were caused by a misinterpretation of the standard by management. One problem noted was the propensity for smaller software development projects to be over-documented. Management must determine what the documentation level of the system is to be as an initial action and must make this decision evident as part of the contract. Documentation complexity and coverage must be determined by considering the projects size, complexity, and planned life-cycle. Items such as the possibility for future modifications, the planned lifetime, and the

criticality of proper performance, must be balanced against the cost and burden of proper documentation. Tradeoffs are inevitable, but a clear decision must be reached in this area.

Once a decision concerning the level and complexity of the documentation for the project has been reached, the decision must be enforced throughout the project's life-cycle. Projects examined exhibited characteristic behavior of early documentation being of high quality and produced on time and within budget; however, as modifications occurred, and time and money limits became a major factor, documentation was quickly put aside in the interest of a fully functional program with the added performance functions. As documentation is an important controlling feature of DOD-STD-1679A, and poor documentation has a "snow-balling" effect as the project moves further down its life-cycle, it is suggested that this not be done so without careful consideration. Perhaps the modification is not so essential, or if it is then every effort should be made to restore the documentation to its high quality level as soon as possible.

The final conclusion was that management does not place a high enough priority on documentation. Although it was not possible to statistically relate project difficulties to inadequate earlier documentation, many of those interviewed expressed a view that their problems would not be as

difficult had they the proper documentation available. The importance of configuration management and quality assurance to a successful project has become well understood. It is suggested that proper documentation, and a continuous effort along that line throughout the life-cycle be elevated to the priorities now enjoyed by CM and QA. This thesis suggested the creation of a documentation group equal in stature to the QA and CM groups to oversee the documentation process. Proper documentation is an investment in the future performance of the software product, and assists in controlling the present.

LIST OF REFERENCES

1. Fairley, R. E., Software Engineering Concepts, McGraw-Hill Book Company, 1985.

2. Boehm, B. W., Software Engineering Economics, Prentice Hall, Inc., 1981.

3. Federal Information Processing Standards Publications 38 and 64, Guidelines for Documentation of Computer Programs and Automated Data Systems, National Bureau of Standards.

4. DOD-STD-1679A(Navy), Military Standard Software Development, p. iii, 1983.

5. DDG 51 Combat System Development, Computer Program Development Plan, Naval Sea Systems Command, p. 1-2, 1985.

6. Mather, D., "One Person's Perception of Military Documentation", AFIPS Conference Proceedings, vol. 53, 1984.

7. NBS Special Publication 500-87, Management Guide for Software Documentation, National Bureau of Standards, p. 12, 1982.

8. Ting, T. C., "An Automated Tool for Program Design and Documentation", Proceeedings of the NBS FIPS Software Documentation Workshop, NBS Special Publication 500-94, p. 95, 1982.

9. O'Conner, P.; Redwine, S. T., "Using FIPS Publication 38: A Practical Experience", Proceedings of the NBS FIPS Software Documentation Workshop, NBS Special Publication 500-94, p. 143, 1982.

INITIAL DISTRIBUTION LIST

	No. Copies
1. Defense Technical Information Center Cameron Station Alexandria, Virginia 22304-6145	2
2. Library, Code 0142 Naval Postgraduate School Monterey, California 93943-5100	2
3. Dr. Gordon H. Bradley, Code 52BZ Department of Computer Science Naval Postgraduate School Monterey, California 93943-5100	2
4. Jim Raegan Naval Surface Weapons Center Code N23 Dhalgren, Virginia 22448	1
5. Dr. Carl R. Jones, Code 54JS Department of Administrative Sciences Naval Postgraduate School Monterey, California 93943-5100	1
6. RADM. Harvey E. Lyon, USN (Ret.) 9210 Bayard Place Fairfax, Virginia 22032	1
7. Lt. Harvey C. Lyon, USN 9210 Bayard Place Fairfax, Virginia 22032	1
8. Computer Technology Programs, Code 37 Naval Postgraduate School Monterey, California 93943-5100	1

END

FILMED

12-85

DTIC